DELPHI

Programmation dans l'environnement Windows

Delphi est un outil de développement, c'est-à-dire un programme qui permet de fabriquer des programmes.

Historique

Plusieurs langages existent et évoluent avec le temps. En voici quelques-uns.

Il y a quelques années, on utilisait :

- Le Basic ou GWBasic
  Ce langage est assez facile à utiliser, mais il génère des applications limitées en taille, assez peu performantes et lentes.
- Le Pascal
  Ce langage est bien structuré, d'une difficulté moyenne, il donne des applications rapides.
- Le C puis C++
  Ce langage laisse une grande liberté au programmeur, il est assez difficile à utiliser, il donne des applications performantes et rapides.

Ces langages généraient des exécutables pour l'environnement DOS.

Avec l'arrivée de Windows, chacun de ces langages a évolué, actuellement on utilise :

- Visual Basic
  Assez facile à utiliser, ce langage génère des exécutables assez lents (cependant, une amélioration de la vitesse vient d'arriver avec la dernière version).
- Delphi
  Ce langage offre beaucoup de possibilités, les exécutables sont très rapides. Il est utilisé par les professionnels. Il existe en version 1 pour Windows 3.x, en version 2 et 3 pour Windows 95 et NT.
- C++ Builder
  Semblable à Delphi, la syntaxe diffère, mais les possibilités sont voisines.

I. Première application avec Delphi

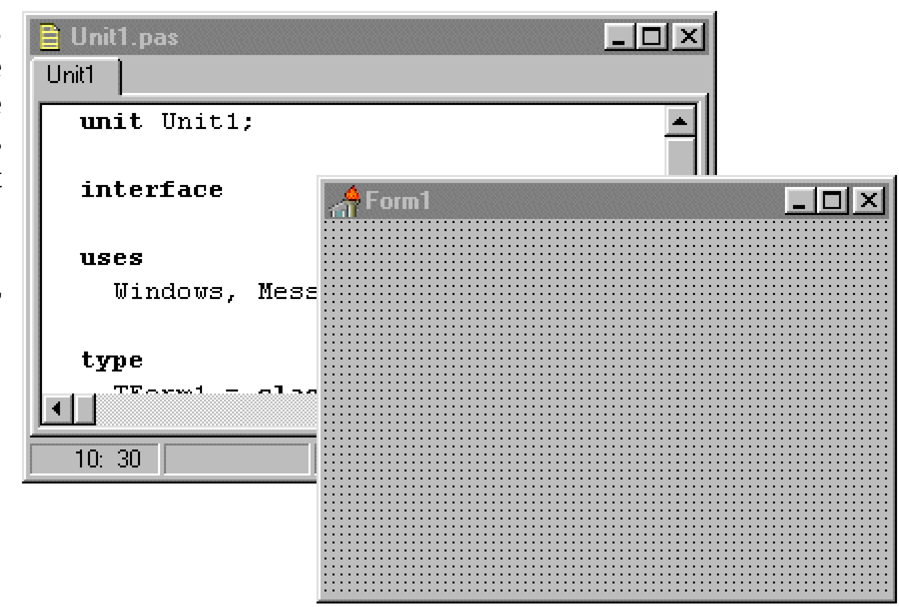
Une application qui ne fait rien

Je vais décrire les quelques étapes qui vont permettre de créer une application :

1. Créez un répertoire dans lequel vous pourrez enregistrer votre travail, l'ordre ne fait pas de mal, surtout en informatique ! Dans la suite, les fichiers ainsi créés seront appelés "le source".

2. Chargez Delphi

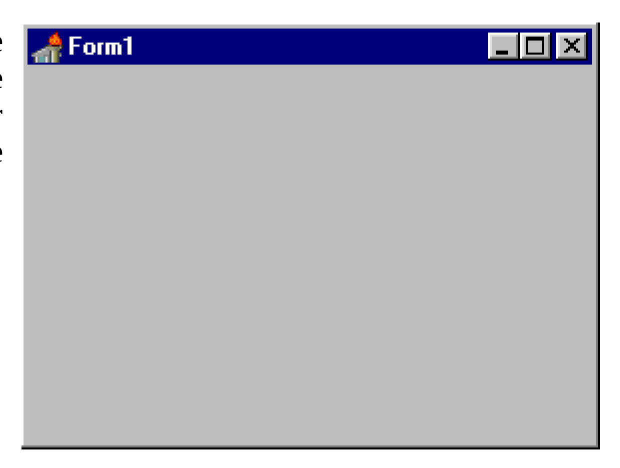
   Si vous n'obtenez pas une fiche vide (nommée Form1), faites "Fichier" et "Nouvelle application". Vous devez obtenir :



3. Faites "Fichier" "Enregistrer sous…" afin d'enregistrer "Unit1.pas" dans le répertoire créé à l'étape 1. Faites également "Fichier" et "Enregistrer projet sous…" afin d'enregistrer "Project1.dpr" dans ce même répertoire.

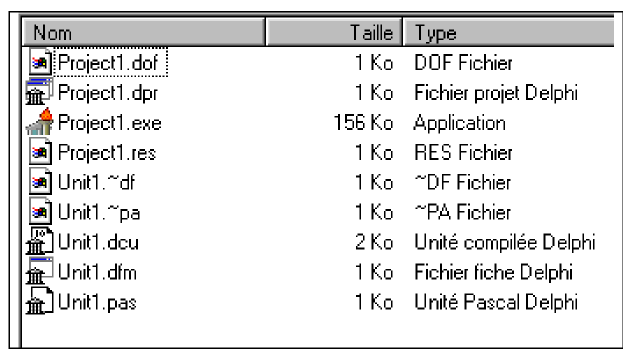
4. Faites "Exécuter" et "Exécuter" (ou ce qui revient au même tapez sur F9, ou encore cliquez sur le bouton "Exécuter"). Delphi compile (c'est-à-dire transforme le code source en un programme exécutable), et exécute ce programme.

   Vous obtenez ainsi une fenêtre nommée "Form1" avec le programme Delphi en arrière-plan. Pour mieux voir votre programme, vous pouvez réduire la fenêtre Delphi.



Votre programme ne fait rien ! c'est une simple fenêtre vide, mais comme avec presque tous les programmes Windows, vous pouvez changer la taille de la fenêtre, la déplacer, cliquer sur le menu système… Si vous quittez votre programme, vous revenez à Delphi.

Plusieurs fichiers ont été enregistrés dans votre répertoire lors de la sauvegarde et de la compilation :



Nom	Taille	Type
Project1.dof	1 Ko	DOF Fichier
Project1.dpr	1 Ko	Fichier projet Delphi
Project1.exe	156 Ko	Application
Project1.res	1 Ko	RES Fichier
Unit1.~df	1 Ko	~DF Fichier
Unit1.~pa	1 Ko	~PA Fichier
Unit1.dcu	2 Ko	Unité compilée Delphi
Unit1.dfm	1 Ko	Fichier fiche Delphi
Unit1.pas	1 Ko	Unité Pascal Delphi

Voici le contenu de "Unit1.pas" mis automatiquement par Delphi :

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Il n'est pas nécessaire de comprendre ces lignes de programme pour faire la suite.

Une fiche avec un bouton

Nous allons améliorer notre programme en ajoutant un bouton sur la fiche.
Si vous ne voyez pas la fiche "Form1", faites "Voir" "Fiches" et choisissez "Form1".
A l'aide de la palette d'outils "Standard", placez un bouton (Button) sur votre fiche. Le bouton comporte des poignées pour le dimensionner. Vous pouvez également déplacer le bouton.

Si vous exécutez le programme, vous verrez la fenêtre nommée "Form1" avec votre bouton. Ce bouton s'enfonce comme tous les boutons des programmes Windows, mais son libellé est peu intéressant et de plus il ne fait rien !

Quittez votre programme pour revenir au mode conception et observez l'inspecteur d'objet (en général placé à gauche de l'écran).

L'inspecteur d'objet

Il comporte deux volets, dans le volet "Propriétés" le mot "Button1" apparaît à deux endroits :
Dans la propriété "Name" vous pouvez modifier le nom de votre bouton. Ce nom sera utilisé par le programme pour faire référence à ce bouton.
Dans la propriété "Caption" vous pouvez modifier le texte qui est écrit sur le bouton.

Dimensionnez éventuellement votre bouton pour que le texte soit entièrement visible. Remarquez qu'en changeant la taille du bouton, les propriétés "Height" et "Width" changent en conséquence. Il était possible de changer ces propriétés au lieu d'utiliser les poignées.
En déplaçant le bouton, ce sont les propriétés "Left" et "Top" qui changent.

Un bouton qui fait quelque chose

Dans la suite, je suppose que votre bouton s'appelle toujours "Button1'.
Vous êtes en mode conception, vous avez le bouton qui est sélectionné.
Dans le volet "Evénements" de l'inspecteur d'objet, faites un double clic sur la zone placée juste à droite de "OnClick".

Delphi vient de vous créer une méthode. Dans notre cas il s'agit d'une procédure affectée à l'événement "OnClick" du bouton.

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Lorsque le programme sera exécuté, et que l'utilisateur fera un clic sur le bouton, Windows déclenchera un événement "OnClick" pour le bouton. Comme cet événement "OnClick" est associé à la méthode "Button1Click" que l'on vient de créer, Windows exécutera cette méthode. Ainsi cette méthode sera exécutée à chaque fois que l'utilisateur fera un clic sur le bouton.

Complétons la méthode à l'aide d'une procédure bien pratique "ShowMessage" qui permet d'afficher une boîte de dialogue nous montrant le message passé en paramètre.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowMessage('Salut à tous');
end;
```

Exécutez le programme, faites un clic sur le bouton (ou tapez sur la barre d'espace lorsque le bouton est activé), le bouton s'enfonce et au moment où il se relève, la boîte de dialogue vous salue.
La boîte de dialogue est modale, cela signifie que vous devez fermer cette boîte avant de pouvoir continuer à exécuter le programme.

A l'aide !

L'aide en ligne est très bien faite, il faut cependant quelquefois un peu d'habitude pour trouver ce que l'on cherche.
Placez le curseur sur le mot ShowMessage et tapez sur F1.

Vous obtenez une explication sur le rôle de cette procédure, la syntaxe à utiliser, les paramètres attendus (dans notre cas une chaîne de caractères).

Vous voyez également que cette procédure fait partie de l'unité "Dialogs". Cela signifie que Delphi ne connaît ce mot que si vous avez "Dialogs" dans la liste des unités utilisées :

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
```

Essayez d'enlever "Dialogs" de la liste et de compiler votre programme, vous obtiendrez une erreur vous indiquant que "ShowMessage" est un "identificateur non déclaré".

II. Notions élémentaires

Variables

Une variable est une zone mémoire. Pour accéder à cette zone mémoire, nous utilisons dans le code que nous tapons, un mot pour l'identifier. Ce mot est le nom de la variable.

Lors de la compilation, Delphi a besoin de savoir quelle quantité de mémoire, il doit réserver. Nous devons commencer à déclarer la variable (à l'aide du mot var) en indiquant son type.

Exemples de déclarations :

```
     var UneVariable : integer;
     var UneAutre : string;
```

Les deux variables ainsi déclarées ne sont pas du même type, la première permet de retenir un nombre entier (positif ou négatif) alors que la deuxième est faite pour retenir une chaîne de caractères (c'est-à-dire une suite de caractères lettres, chiffres ou signes…).

Pour mettre une valeur dans une variable, il faut indiquer :

A gauche	le nom de la variable
Suivi du symbole	**:=**
A droite	une valeur du bon type (sous forme simple, ou sous forme à calculer).

Exemples d'affectation :

```
     UneVariable := 123;
     UneVariable := -20549;
     UneVariable := 15+17*3;
     UneAutre := 'Etude de la programmation avec Delphi.';
```

Où peut-on placer ces lignes de programme ?
Les déclarations peuvent être placées dans une procédure ou une fonction, avant le mot **begin**.
Les affectations peuvent être placées dans une procédure ou une fonction, après le mot **begin** et avant le mot **end;**.

Nous verrons plus tard que d'autres places sont possibles.

Vous pouvez par exemple modifier la procédure Button1Click comme suit :

```
procedure TForm1.Button1Click(Sender: TObject);
var UneVariable:integer;
var UneAutre:string;
begin
   UneVariable:=53;
   UneAutre:='Bonjour, ceci est une chaîne.';
   ShowMessage(UneAutre);
end;
```

Remarques :

- *N'oubliez pas le point-virgule, il est nécessaire après chaque instruction pour la séparer de la suivante.*
- *La variable nommée UneVariable ne sert à rien, elle n'est jamais utilisée, aussi Delphi ignore tout ce qui concerne UneVariable lors de la compilation afin d'optimiser le code dans le fichier exécutable créé.*

Une boucle

Une boucle permet de répéter une partie d'un programme un certain nombre de fois.

La syntaxe de la boucle est

for *nom de variable entière* **:=** *première valeur* **to** *dernière valeur* **do**
begin
ce qu'il faut faire plusieurs fois
end;

Pour mettre ceci en évidence sur un exemple, je vous propose de créer un deuxième bouton, (je suppose que vous l'appelez "ButtonBoucle" et que vous mettez "Essai de boucle" dans sa propriété Caption.
Faites un double clic sur le bouton et complétez la procédure obtenue comme suit :

```
procedure TForm1.ButtonBoucleClick(Sender: TObject);
var i : integer;
var S : string;
begin
   for i:=1 to 5 do
   begin
      S := IntToStr(i);
      ShowMessage('Message n°' + S);
   end;
end;
```

En exécutant le programme et en cliquant sur le bouton "Essai de boucle", vous obtenez 5 messages successifs "Message n°1" puis "Message n°2" … jusqu'à "Message n°5".

La fonction IntToStr permet de convertir le nombre placé dans la variable i en une chaîne.
Lorsque la variable i contient par exemple le nombre 3, la variable S reçoit la chaîne '3'.
Les deux chaînes 'Message n°' et S sont mises l'une au bout de l'autre à l'aide du signe +. On dit que les chaînes ont été concaténées.

Débogage

Placez le curseur sur la ligne commençant par for et tapez sur F5 (on peut aussi faire "Ajouter point d'arrêt" à l'aide des menus).
Exécutez le programme et cliquez sur le bouton "Essai de boucle".
Le programme s'arrête sur la ligne contenant le point d'arrêt.

Utilisez F8 ou F7 pour avancer en "pas à pas".

Pour observer le contenu d'une variable :
Avec Delphi version 3, il suffit de placer (sans cliquer) le pointeur souris sur le nom de la variable et d'attendre une seconde.
Avec toutes les versions, placez le curseur sur la variable et faites CTRL et F7 (ou utilisez "Evaluer/modifier" dans les menus).

Si vous obtenez le message "La variable … est inaccessible ici du fait de l'optimisation" vous pouvez dans "Projet" "Option" "Compilateur" supprimer la coche dans "Optimisation". L'exécutable obtenu sera un peu moins performant, mais vous pourrez voir toutes les variables dès la prochaine compilation.

Pour continuer l'exécution du programme à la vitesse normale, vous pouvez taper sur F9.

Comment supprimer un composant

Pour supprimer par exemple Button1, vous allez commencer par vider toutes les méthodes liées aux événements de Button1. Dans notre cas, seul l'événement "OnClick" déclenche une procédure. Supprimez les lignes situées entre la ligne "Procedure …" et la ligne "Begin" ainsi que les lignes entre Begin et End;

Votre procédure doit devenir :

```
procedure TForm1.Button1Click(Sender: TObject);
begin
end;
```

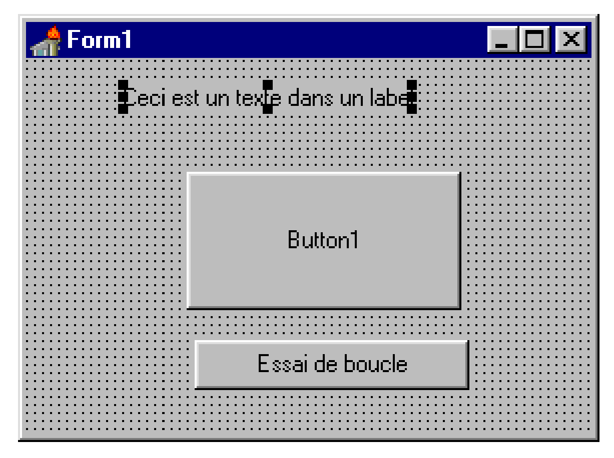
Tapez sur F12 pour voir la fiche et supprimez le bouton "Button1"

A la prochaine sauvegarde ou à la prochaine compilation, les lignes inutiles seront automatiquement supprimées.

Label

Le composant "Label" permet d'écrire du texte dans une partie de la fiche.

Dans la palette "Standard" des composants choisissez le composant "Label" représenté par un bouton marqué **A**. Placez un label sur la fiche et mettez une phrase dans sa propriété "Caption". La phrase tapée pour l'exemple ci-après est "Ceci est un texte dans un label"



Conception et exécution

Une propriété peut souvent être modifiée lors de la conception du programme et par le programme lorsqu'il est exécuté. Pour modifier le texte (Caption) du label pendant l'exécution du programme, il suffit d'ajouter la ligne :

```
   Label1.Caption:='Le texte que l''on veut voir apparaître';
```

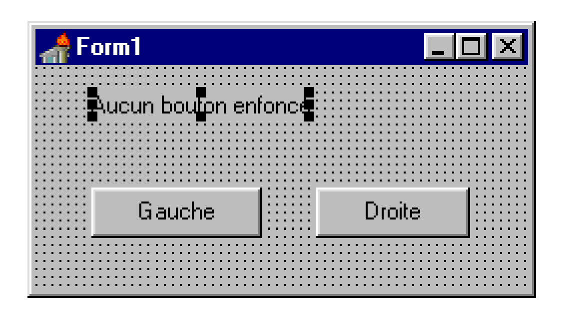
Remarque : quand, dans une chaîne de caractères, vous avez besoin d'une apostrophe, il suffit d'en mettre deux de suite pour que Delphi comprenne qu'il ne s'agit pas de la fin de la chaîne.

Pour mettre ceci en évidence, je vous propose de faire "Fichier" et "Nouvelle application"

Gardons les bonnes habitudes :
Faites tout de suite "Enregistrer sous", créez un nouveau répertoire et enregistrez Unit1.pas (en changeant éventuellement son nom) dans ce nouveau répertoire.

Faites "Enregistrer Projet sous" et enregistrez Project1 (en changeant éventuellement son nom) dans ce même répertoire.

Placez un label, nommez-le par exemple LabelInformation, mettez "Aucun bouton enfoncé" dans sa propriété "Caption".



Placez deux boutons l'un à gauche et l'autre à droite, nommez par exemple celui de gauche "ButtonGauche" et celui de droite "ButtonDroite", Mettez dans la propriété Caption du bouton de gauche le mot "Gauche" et dans celle du bouton de droite le mot "Droite".

Notre but est que le texte dans le label indique suivant le cas :

"Le bouton de gauche vient d'être enfoncé"

ou

"Le bouton de droite vient d'être enfoncé".

Il nous reste à concevoir les procédures qui seront exécutées lors des événements "OnClick" de chaque bouton. Voici le code que vous devriez obtenir.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    LabelInformation : TLabel;
    ButtonGauche : TButton;
    ButtonDroite : TButton;
    procedure ButtonGaucheClick(Sender: TObject);
    procedure ButtonDroiteClick(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.ButtonGaucheClick(Sender: TObject);
begin
   LabelInformation.caption:='Le bouton de gauche vient d''être enfoncé';
end;

procedure TForm1.ButtonDroiteClick(Sender: TObject);
begin
   LabelInformation.caption:='Le bouton de droite vient d''être enfoncé';
end;
```

```
end.
```

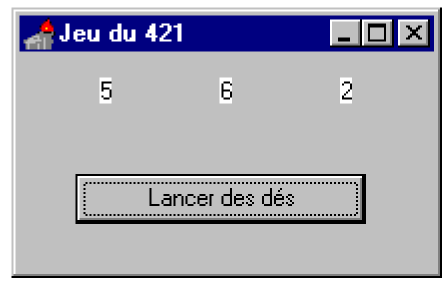
Vous ne devez rien taper dans la partie située entre la ligne class et la ligne private, c'est Delphi qui se charge de modifier ces lignes en fonction des composants que vous placez dans la fiche.
Pour créer les procédures, souvenez-vous qu'il suffit de faire un double clic sur le bouton correspondant.

Exécutez le programme et observez le texte du Label avant d'avoir fait un clic sur un bouton et après.

Exercice 1

A l'aide de trois labels et d'un bouton, simulez le lancer de trois dés.

Vous aurez besoin de la procédure "Randomize" et de la fonction "Random". Cherchez ces mots dans l'aide.
Après un clic sur le bouton de ce programme vous obtiendrez par exemple :



Pour mieux mettre en évidence les trois labels, j'ai changé leur propriété "color" en choisissant "clWhite".
Le titre "Jeu du 421" de la fenêtre a été mis en mode conception en activant la fiche (en cliquant sur la fiche mais pas sur un label ou sur le bouton) et en changeant la propriété "Caption" de "Form1".

Et si vous avez de la chance vous obtiendrez 421 !

Ne regardez le code à placer dans Button1Click qu'après avoir cherché !

```
randomize;
Label1.Caption:=IntToStr(random(6)+1);
Label2.Caption:=IntToStr(random(6)+1);
Label3.Caption:=IntToStr(random(6)+1);
```

III. Des types

Généralités sur les types

Lorsque vous déclarez une variable, le compilateur a besoin de savoir quelle place il doit lui réserver en mémoire. Il ne faut en effet pas la même place pour retenir un nombre entier et une chaîne.
Avec Delphi version 2 ou 3, il faut 4 octets pour retenir un Integer, alors qu'un String nécessite 9 octets en plus des caractères composant la chaîne.

De plus Delphi a besoin de connaître les opérations qu'il peut effectuer avec la variable. Il peut multiplier des entiers mais pas des chaînes !

Un type est une sorte de modèle de variable mais ce n'est pas une variable. Un type ne prend pas de place en mémoire, il permet seulement au compilateur de savoir quelle place il faut prévoir pour chaque variable déclarée avec ce type et d'interdire toute opération qui ne conviendrait pas pour ce type.

Par exemple

```
Type Str80 = string[80];
```

signifie que nous venons de définir un type nommé Str80 qui nous permettra de retenir une chaîne de 80 caractères maximum. Aucune place mémoire n'est allouée à ce stade.

La déclaration de type peut être placée dans la partie "interface" ou dans la partie "implementation" ou encore au début (avant le begin) d'une procédure ou d'une fonction.

En ajoutant les lignes

```
Var S1 : Str80;
Var S2 : Str80;
```

Delphi réserve la mémoire suffisante pour retenir 80 caractères pour S1 et 80 caractères pour S2.

Les déclarations des variables peuvent être placées à plusieurs endroits mais pour l'instant, on se contentera de les placer au début d'une procédure ou d'une fonction avant le mot begin.

La ligne :

```
S1:=123;
```

causera une erreur lors de la compilation.

Alors que la ligne :

```
S1:='Bonjour à tous';
```

est correcte.

L'instruction d'affectation d'une valeur à une variable peut être placée à plusieurs endroits mais pour l'instant, on se contentera de la placer entre le begin et le end d'une procédure ou d'une fonction.

Les types numériques

Les deux types numériques les plus utilisés sont

Integer

Et

Real

Dans le premier on est limité à des nombres entiers de -2147483648 à 2147483647.
Dans le deuxième les nombres à virgule sont admis et la plage va de $2.9 ? 10^{-39}$ à $1.7 ? 10^{38}$ avec des valeurs positives ou négatives.

Pour en savoir plus, regardez dans l'index de l'aide les pages "Types entiers" et "Types réels"

Les types chaînes

Les chaînes courtes (255 caractères maximum) peuvent être du type **ShortString**.
Depuis la version 2 de Delphi, le type **String** permet de manipuler des chaînes de toute taille. La place allouée pour une variable de type String change automatiquement lorsque la chaîne change de taille.

Le type boolean

Le type **boolean** ne permet que deux états. Une variable de type **boolean** ne peut valoir que <u>true</u> ou <u>false</u>.

Les types tableaux

Un tableau permet de retenir un ensemble de données de même type.
Il faut préciser après le mot **array** le premier et le dernier indice du tableau.

```
Type TMonTableau = array[5..8] of integer;
```

Quand le compilateur rencontrera ces lignes, il n'allouera pas de mémoire.
Remarquez le T permettant de se souvenir qu'il s'agit d'un type. C'est pratique, c'est une bonne habitude mais ce n'est pas obligatoire.

```
Var MonTableau : TMonTableau;
```

Quand le compilateur rencontrera cette ligne, il allouera la mémoire nécessaire.

Les deux lignes de code précédentes pouvaient se réduire à une seule :

```
Var MonTableau : array[5..8] of integer;
```

Le tableau ainsi créé permet de retenir 4 nombres entiers. Voici des exemples de lignes de programme correctes :

```
MonTableau[5] := 758;
MonTableau[6] := MonTableau[5] * 2;
MonTableau[7] := -123;
MonTableau[7] := MonTableau[7] + 1;
MonTableau[8] :=MonTableau[5] - MonTableau[7];
Label1.Caption := IntToStr(MonTableau[8]);
```

La dernière ligne suppose que la fiche contient un label nommé Label1.

Question : Quel nombre sera affiché dans le label ? (ne regardez pas la réponse sans avoir cherché).

Le nombre 880

Il est souvent plus commode de commencer les indices de tableau à 1 ou à 0. Il est possible de créer des tableaux à plusieurs dimensions. Exemples

```
Var NotesConcours : array[1..12] of Real;
Var NotesPromo : array[1..100,1..12] of real;
```

La première variable tableau permettrait de retenir le résultat d'un candidat à un concours comportant 12 épreuves.
La deuxième variable tableau permettrait de retenir le résultat de 100 candidats à un concours comportant 12 épreuves.

Pour indiquer que le candidat numéro 25 a eu 13,5 à l'épreuve numéro 4, on pourra écrire :

```
NotesPromo[25,4] := 13.5;
```

Pour connaître le résultat du candidat numéro 25 à l'épreuve numéro 4, on pourra écrire :

```
Label1.Caption:='Le candidat a eu ' + IntToStr(NotesPromo[25,4]);
```

Pour en savoir plus sur les tableaux regardez dans l'index de l'aide "Types tableau".

Exercice 2

Faire un programme qui remplit (de façon fixe ou aléatoire) les 12 notes dans le tableau NotesConcours et qui, à l'aide d'une boucle, calcule la moyenne du candidat. Faire apparaître cette moyenne dans un label.

Les types enregistrements

Ne pas confondre avec l'enregistrement d'un fichier sur le disque, il s'agit ici d'un regroupement de types plus simples.

```
Type TUnePersonne = record
   Nom : string[25];
   Prenom : string[25];
   Age : ShortInt;
End;
```

Quand le compilateur rencontrera ces lignes, il n'allouera pas de mémoire.

```
Var UnePersonne:TUnePersonne;
```

Quand le compilateur rencontrera cette ligne, il allouera dans la mémoire une zone d'au moins 51 octets.

```
UnePersonne.Nom := 'DUPOND';
UnePersonne.Prenom := 'Albert';
UnePersonne.Age := 35;
```

Quand le compilateur rencontrera ces lignes, il remplira les zones mémoires précédemment allouées.

Pourquoi pas un tableau d'enregistrements ?

```
Type TTableauPersonnes = array[1..50] of TUnePersonne;
```

```
Var TableauPersonnes : TTableauPersonnes;
```

Pour indiquer par exemple que la troisième personne s'appelle DUBOIS Cathy et qu'elle a 18 ans, on pourra taper :

```
TableauPersonnes[3].Nom := 'DUBOIS';
TableauPersonnes[3].Prenom := Cathy';
TableauPersonnes[3].Age := 18;
```

Pour copier cette personne à la première place du tableau, on pourra au choix faire :

```
TableauPersonnes[1].Nom := TableauPersonnes[3].Nom;
TableauPersonnes[1].Prenom := TableauPersonnes[3].Prenom;
TableauPersonnes[1].Age := TableauPersonnes[3].Age;
```

Ou plus simplement :

```
TableauPersonnes[1] := TableauPersonnes[3];
```

IV. Portée des variables

La place où est déclarée une variable est importante.

Variable locale à une procédure ou une fonction

Les variables que nous avons déclarées précédemment l'ont toujours été à l'intérieur d'une procédure ou d'une fonction. La portée (la partie du programme où la variable est utilisable) est la procédure ou la fonction. Si, par exemple nous déclarons une variable Ch dans une procédure, nous ne pouvons pas utiliser cette variable dans une autre procédure. En fait, lorsque votre programme sera exécuté et que cette procédure sera appelée, de la mémoire sera automatiquement allouée pour cette variable. La procédure pourra donc l'utiliser librement (mettre quelque chose dedans, lire son contenu…). Dès que la procédure sera terminée, la place allouée pour cette variable sera automatiquement libérée.

Quand la procédure sera à nouveau exécutée, la variable sera à nouveau créée, il ne faut donc pas espérer retrouver la valeur précédente. La valeur est même tout à fait imprévisible !

Rien ne vous empêche de déclarer des variables de même nom dans des procédures ou des fonctions différentes, mais aucun lien ne sera établi entre ces variables.

Variable définie dans la classe

Pour qu'une variable soit utilisable par toutes les méthodes de la classe, on placera la déclaration dans la partie **class** après le mot **private** ou après le mot **public**. Dans ce cas, on ne précise pas le mot **var**.

Voici un exemple où j'ai ajouté deux variables NomEntreprise et NombreVehicules dans la partie **private** de la déclaration du type **class** de TForm1.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    ButtonAjoute: TButton;
    ButtonEnleve: TButton;
  private
    { Déclarations privées }
    NomEntreprise : string;
    NombreVehicules : integer;
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;
```

J'ai créé une nouvelle application, l'ai enregistrée dans un répertoire créé pour elle. J'ai placé un label et deux boutons, et enfin j'ai ajouté les deux variables.

Les procédures qui seront exécutées lorsque que l'on fera un clic sur l'un des boutons sont des méthodes de la même classe (la classe TForm1, la seule que nous avons vue pour l'instant), elles pourront donc toutes les deux utiliser nos deux variables.
D'une façon générale, toute méthode de TForm1 déclarée dans le type classe pourra utiliser nos deux variables.

Voici le programme complété par des méthodes (procédures) utilisant les deux variables :

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    ButtonAjoute: TButton;
    ButtonEnleve: TButton;
    procedure FormCreate(Sender: TObject);
    procedure ButtonAjouteClick(Sender: TObject);
    procedure ButtonEnleveClick(Sender: TObject);
  private
    { Déclarations privées }
    NomEntreprise : string;
    NombreVehicules : integer;
    procedure MetDansLeLabel;
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.MetDansLeLabel;
begin
  Label1.Caption:='L''entreprise ' + NomEntreprise +
                  ' possède ' + IntToStr(NombreVehicules) + ' véhicule(s)';
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  NomEntreprise:='SOCIETE DE TRANSPORT';
  NombreVehicules:=0;
  MetDansLeLabel;
end;

procedure TForm1.ButtonAjouteClick(Sender: TObject);
begin
  NombreVehicules := NombreVehicules + 1;
  MetDansLeLabel;
end;

procedure TForm1.ButtonEnleveClick(Sender: TObject);
begin
  NombreVehicules := NombreVehicules - 1;
  MetDansLeLabel;
end;
```

```
end.
```

Des remarques à propos de ce listing.

- Les variables NomEntreprise et NombreVehicules sont utilisées dans plusieurs procédures.
- La procédure FormCreate a été placée de façon automatique dans le source en cliquant sur la fiche (pas sur le label ni sur un bouton), l'inspecteur d'objet doit afficher les propriétés de Form1. On a ensuite activé l'onglet 'Evénements' et fait un double clic à droite de "OnCreate".
- La procédure FormCreate sera exécutée une fois lorsque le programme démarrera, avant même que la fiche ne soit visible.
- Les trois lignes de programme dans le corps de la procédure FormCreate ont été tapées normalement.
- On a tapé une ligne dans la partie **private** pour déclarer la procédure MetDansLeLabel.
- Dans la partie **implementation**, on a tapé la procédure MetDansLeLabel en faisant précéder son nom de TForm1 et d'un point pour montrer qu'il s'agit d'une procédure déclarée dans la classe TForm1.
- On peut remarquer que la ligne de programme dans le corps de la procédure MetDansLeLabel utilise deux lignes de l'éditeur. C'est une possibilité afin d'éviter les lignes trop longues en particulier si on imprime le listing. C'est le point-virgule qui détermine la fin de l'instruction.
- Lorsque le programme sera exécuté, en cliquant sur le bouton ButtonAjouter la valeur dans la variable NombreVehicules sera augmentée de 1 et la procédure MetDansLeLabel sera exécutée.
- De même pour BoutonEnlever en diminuant de 1.
- Au lieu de taper NombreVehicules := NombreVehicules +1; on aurait également pu taper inc(NombreVehicules);
- De même au lieu de NombreVehicules := NombreVehicules -1; on aurait pu taper dec(NombreVehicules);
- Il n'y a pas de contrôle sur le nombre de véhicules, il peut devenir négatif !

Si vous utilisez Delphi 3, au lieu de taper F9 pour exécuter le programme, je vous conseille d'essayer le pas à pas approfondi en tapant F7 afin d'exécuter les instructions une par une. Vous verrez ainsi l'exécution de la procédure FormCreate puis en cliquant sur un bouton, vous verrez (toujours en tapant plusieurs fois sur F7) les lignes de la procédure correspondante.

Avec Delphi 1 ou 2 vous devrez placer des points d'arrêt au début de chaque procédure que vous voulez étudier en pas à pas. Tapez alors F9, cliquez éventuellement sur un bouton puis, utilisez F7 lorsque le programme s'arrête sur votre point d'arrêt.

D'autres places sont possibles

Il est possible de placer la déclaration de variables en dehors des endroits étudiés précédemment mais c'est pour une utilisation spéciale et il est préférable dans presque tous les cas de ne pas le faire.

V. Procédures

Procédure sans paramètre

Dans le chapitre précédent, la méthode MetDansLeLabel est une procédure qui évite de répéter plusieurs fois les mêmes lignes. En effet, en l'absence de cette procédure, il aurait fallu taper 3 fois la même ligne pour mettre à jour le libellé du label. Cette procédure ne possède pas de paramètre.

Procédure avec paramètre

Reprenons l'exemple de l'entreprise et des véhicules. Au lieu d'ajouter 1 ou de retrancher 1, nous allons également donner la possibilité d'ajouter 5 ou de retrancher 5.
Pour cela, ajoutons deux autres boutons ButtonAjouter5 et ButtonEnlever5.

Nous allons donner le soin à la procédure MetDansLeLabel de modifier la valeur de NombreVehicules. Modifions-la de la façon suivante :

Dans la définition de la classe :

```
procedure MetDansLeLabel( DeCombien : integer );
```

Dans la partie implémentation :

```
procedure TForm1.MetDansLeLabel( DeCombien : integer );
Begin
  NombreVehicules : =NombreVehicules + DeCombien;
  Label1.Caption:='L''entreprise ' + NomEntreprise +
                  ' possède ' + IntToStr(NombreVehicules) + ' véhicule(s)';
End;
```

La procédure liée à l'événement OnClick du bouton ButtonAjoute5 sera :

```
procedure TForm1.ButtonAjoute5Click(Sender: TObject);
Begin
  MetDansLeLabel(5)
End;
```

Et celle liée à l'événement OnClick du bouton ButtonEnlever5 sera :

```
procedure TForm1.ButtonAjoute5Click(Sender: TObject);
Begin
  MetDansLeLabel(-5)
End;
```

Les procédures liées à l'événement "OnClick" des deux premiers boutons devront être également modifiées et ne contiendront plus que MetDansLeLabel(1) ou MetDansLeLabel(-1).

Plusieurs paramètres sont possibles

Dans la déclaration de la procédure, les variables doivent être séparées par un point-virgule. Quand on appelle la procédure, les valeurs passées doivent être séparées par une virgule.

Par exemple la procédure déclarée :

```
procedure TForm1.Essai( Nbr:integer ; S : string ; Action : integer);
begin
  // mettre des lignes de programme qui peuvent utiliser les variables
end;
```

pourra être utilisée en passant trois paramètres du bon type et dans le bon ordre :

```
Essai( 25 * 2 , 'Un message quelconque' , 2);
```

Passage de paramètre par valeur ou par référence

```
procedure TForm1.EtudeParametre( A : integer ; var B : integer);
begin
  ShowMessage('Début de EtudeParametre A=' + IntToStr(A) + ' B=' + IntToStr(B));
  A := A + 5;
  B := B + 5;
  ShowMessage('Fin de EtudeParametre A=' + IntToStr(A) + ' B=' + IntToStr(B));
end;
   :
procedure TForm1.Button1Click(Sender: Tobject);
var X:integer;
    Y:integer;
begin
  X := 100;
  Y := 4000;
  ShowMessage('Avant EtudeParametre X=' + IntToStr(X) + ' Y=' + IntToStr(Y));
  EtudeParametre(X , Y);
  ShowMessage('Après EtudeParametre X=' + IntToStr(X) + ' Y=' + IntToStr(Y));
end;
```

Remarquez le mot **var** devant la variable B.

En cliquant sur le bouton Button1, quatre messages successifs vont s'afficher.
- Le premier montre les valeurs qui viennent d'être mises dans les variables X et Y.
- Le deuxième et le troisième étant dans la procédure EtudeParametre, ils ne peuvent montrer que les variables A et B.
- Le quatrième montre que la variable X n'a pas été modifiée alors que Y est modifiée.

La <u>valeur</u> de X a été attribuée à la variable A de EtudeParametre. Le passage s'est fait par <u>valeur</u>.

L'<u>adresse</u> de la variable Y a été donnée à la procédure EtudeParametre, et tout changement de la valeur de la variable B change en fait la même zone mémoire que celle attribuée à Y. Le passage s'est fait par <u>référence</u>.

VI. Tests

If then

if *expression qui peut être vraie ou fausse* **then**
begin
// ce qu'il faut faire si vrai
end;
// suite faite dans tous les cas

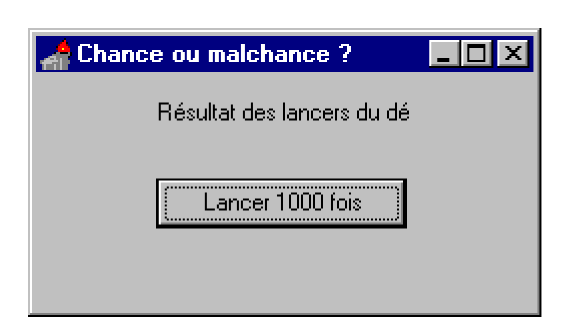
Si, par exemple, vous devez gérer le stock d'un magasin, vous serez amené à contrôler le nombre d'articles afin de décider à quel moment il faudra passer commande. La partie du programme pourrait contenir :

```
if NbrSacsCimentEnStock <50 then
  begin
    EnvoyerCommande('Sacs de ciment',120);
  end;
```

Bien sûr ces lignes supposent que la variable NbrSacsCimentEnStock soit définie et que sa valeur reflète effectivement le stock. Elles supposent également que la procédure EnvoyerCommande existe et qu'elle demande un paramètre chaîne puis un paramètre entier.

Reprenons un exemple qui utilise le hasard. Nous allons lancer un dé par exemple 1000 fois et nous allons comptabiliser le nombre de fois que l'on obtient le 6.
Nous devrions en général obtenir aux environs de 167, mais rien n'interdit en théorie d'obtenir 1000 fois le 6 ou de ne jamais le voir !

Dans une nouvelle application, placez un label et un bouton sur la fiche.
En cliquant sur le bouton, une boucle va simuler les 1000 lancers du dé, le label nous permettra de voir le résultat. Le test sera fait à l'intérieur de la boucle (entre le begin et le end correspondant au for).
Essayez de faire ce programme puis comparer avec ce qui suit.




```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
```

```
    LabelResultat: TLabel;
    ButtonLancer: TButton;
    procedure ButtonLancerClick(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.ButtonLancerClick(Sender: TObject);
var i:integer;
var Nbr6:integer;
begin                                               {début de la procédure}
  randomize;
  Nbr6:=0;
  for i := 1 to 1000 do
  begin                                             {début de la boucle}
    if random(6)+1 = 6 then
    begin                                           {début du test}
      inc(Nbr6);
    end;                                            {fin du test}
  end;                                              {fin de la boucle}
  LabelResultat.Caption:='Nombre de 6 : ' + IntToStr(Nbr6);
end;                                                {fin de la procédure}

end.
```

Il est nécessaire de mettre 0 dans la variable Nbr6 avant de commencer la boucle, en effet, à l'entrée dans une procédure ou une fonction, les variables déclarées localement ont une valeur quelconque.

Essayez de supprimer la ligne Nbr6 := 0 et vous obtiendrez des résultats certainement farfelus.

Essayez de ne pas déclarer la variable Nbr6 dans la procédure mais de la déclarer dans la partie classe (entre le mot private et le mot public). Ne mettez pas la ligne Nbr6 : = 0.

La première fois que vous cliquez sur le bouton, la variable Nbr6 commence à 0, vous obtenez un nombre de 6 raisonnable, la deuxième fois, il y a cumul avec les lancers de la première fois… Si vous arrêtez et lancez à nouveau le programme, Nbr6 recommence à 0.

Une variable déclarée dans la partie classe est mise automatiquement à 0 pour les variables numériques, à false pour les variables booléennes et à chaîne vide pour les variables chaînes.

If then else

Il arrive assez souvent qu'une action doive être faite si le test est vrai et une autre si le test est faux.

if *expression qui peut être vraie ou fausse* **then**
begin
  // ce qu'il faut faire si vrai
end
else
begin
  // ce qu'il faut faire si faux
end;
 // suite faite dans tous les cas

En supposant que Moy ait été définie comme **real** et qu'elle contienne le résultat d'un examen, une partie du programme pourrait être :

```
if Moy >= 10 then
begin
  Label1.Caption := 'Examen réussi';
end
else
begin
  Label1.Caption := 'Echec';
end;
```

Pour indiquer "supérieur ou égal" on utilise > suivi de =.
L'expression Moy >= 10 peut être vraie (true) ou fausse (false)

Exemple récapitulatif

Nous allons fabriquer une procédure qui transforme un nombre entier en un nombre en base 16. Quelques exemples pour se familiariser avec la base 16 :

Les nombres	se décomposent en	s'écrivent en hexadécimal
de 0 à 9		0 à 9
10 à 15		A à F
16		10 (lire un zéro)
25	16 + 9	19 (lire un neuf)
56	3*16 + 8	38 (lire trois huit)
3135	12*16*16 + 3*16 + 15	C3F

Pour décomposer un nombre en base 16, nous devons faire une division entière par 16, mettre le reste comme dernier chiffe hexadécimal, recommencer avec le quotient etc.

3135 divisé par 16 donne 195 et le reste est 15, le dernier chiffre hexadécimal est donc F.
195 divisé par 16 donne 12 et le reste est 3, l'avant dernier chiffre hexadécimal est donc 3.
12 donne C comme premier chiffre.

Nous allons créer une procédure qui aura pour rôle d'afficher, dans un Label, la valeur hexadécimale d'un nombre qui lui est passé comme paramètre. Appelons EntierVersHexa cette procédure.

```
procedure TForm1.EntierVersHexa( Nbr : integer );
var i : integer;
    Quotient : integer;
    Reste : integer;
    H : string;
begin
  H := '';
  for i := 1 to 4 do
  begin
    Quotient := Nbr div 16;                  { div effectue une division entière }
    Reste : =Nbr - Quotient * 16;             { Reste vaut entre 0 et 15 }
    if Reste <= 10 then
    begin
      H := chr(Reste + 48) + H;               { Reste + 48 vaut entre 48 et 57 }
    end
    else
    begin
      H := chr(Reste + 55) + H;               { Reste + 55 vaut entre 65 et 70 }
    end;
    Nbr := Quotient;
  end;
  Label1.Caption := H;
end;
```

Chr(48) vaut le caratère '0', chr(49) le caractère '1' … chr(57) le caractère '9'.
Chr(65) vaut le caratère 'A', chr(66) le caractère 'B' … chr(70) le caractère 'F'.
Si Reste vaut 9, alors Reste + 48 vaut 57 ce qui correspond au caractère '9'.
Si Reste vaut 10, alors Reste + 55 vaut 65 ce qui correspond au caractère 'A'.

Pour tester cette procédure, vous pouvez l'appeler à partir d'une méthode Button1Click d'un bouton par exemple en tapant :

```
  EntierVersHexa( 3135 );
```

Remarque : L'exercice ne présente qu'un intérêt pédagogique, en effet Delphi fournit une fonction nommée IntToHex qui fait la même conversion.

VII. TEdit permet une saisie de l'utilisateur

Jusqu'à présent, l'utilisateur n'avait pas d'autre choix que de faire un clic sur un bouton, nous allons maintenant permettre à l'utilisateur de taper une chaîne de caractères.

Afficher le double (avec un bouton)

Ce programme va se contenter de permettre à l'utilisateur de taper un nombre et, lorsqu'il fera un clic sur un bouton, il verra s'afficher le double du nombre tapé. Bien sûr il serait plus intéressant d'effectuer une opération plus complexe et plus utile, mais il s'agit ici de voir le principe.

Plaçons dans un nouveau programme, un composant de type "TEdit", un de type "TLabel" et un de type "TButton". Si vous ne changez pas leur nom, ils s'appelleront respectivement Edit1, Label1 et Button1.

Le composant Edit1 permet à l'utilisateur de taper une chaîne de caractères, il faut la convertir en un nombre, doubler le nombre et convertir le résultat en une chaîne de caractères pour la placer dans la propriété Caption de Label1.

Un composant TEdit contient le texte dans sa propriété Text, on y accède par Edit1.Text.

Voir dans l'aide le mot StrToInt.

```
procedure TForm1.Button1Click(Sender: Tobject);
var Nbr : integer;
begin
  Nbr := StrToInt(Edit1.Text);
  Nbr := Nbr * 2;
  Label1.Caption := IntToStr(Nbr);
end;
```

Essayez le programme en mettant des nombres dans la zone d'édition, essayez également en mettant autre chose qu'un nombre entier.

En cas d'erreur, Delphi déclenche une exception. L'utilisateur est informé de l'erreur et la suite de la procédure est ignorée. Le programme n'est pas arrêté, l'utilisateur peut donc faire de nouveaux essais.

Le double si possible (avec un bouton)

Nous pouvons limiter les erreurs en n'autorisant à l'utilisateur que la frappe des chiffres, mais s'il tape un nombre trop grand, une erreur se produira malgré tout. Une meilleure solution consiste à utiliser les mots **try** et **except**.

Voyons comment mettre en place la première solution.
Nous allons autoriser les 10 chiffres, Key peut donc valoir entre '0' et '9'.
Nous allons également autoriser la touche "retour arrière" pour permettre la correction. Cette touche correspond au code 8. Il faut donc autoriser Key=chr(8).

Toutes les autres touches tapées dans le composant Edit1 seront refusées.

Dans les événements de Edit1, faites un double clic à côté de "OnKeyPress", vous obtenez la méthode Edit1KeyPress.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key <> chr(8) then
  begin
    if Key < '0' then
    begin
      Key := chr(0);              { permet d'annuler la frappe de la touche }
    end;
    if Key > '9' then
    begin
      Key := chr(0);              { permet d'annuler la frappe de la touche }
    end;
  end;
end;
```

Autre solution en utilisant les mots and et or :

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if (Key <> chr(8)) and ((Key < '0') or (Key > '9')) then
  begin
    Key := chr(0);              { permet d'annuler la frappe de la touche }
  end;
end;
```

La variable Key est passée par référence (par adresse) ; si nous la modifions, elle sera également modifiée en dehors de la procédure.

Le double si possible et tout de suite (sans bouton)

A chaque changement du contenu de Edit1, nous allons mettre à jour le texte affiché dans Label1. C'est la méthode liée à l'événement "OnChange" de Edit1 qui va faire ce travail.

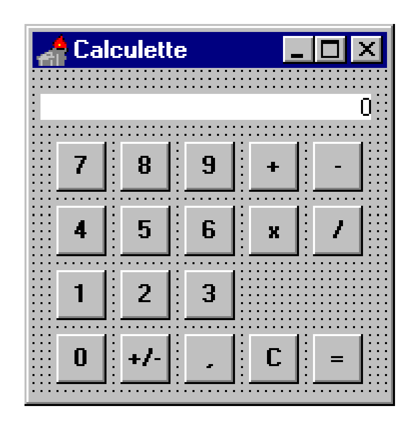
Déplacez le code qui était dans la procédure TForm1.Button1Click et mettez-le dans TForm1.Edit1Change. Vous pouvez supprimer le bouton Button1.

```
procedure TForm1.Edit1Change(Sender: TObject);
var Nbr : integer;
begin
  Nbr := StrToInt(Edit1.Text);
  Nbr := Nbr * 2;
  Label1.Caption := IntToStr(Nbr);
end;
```

Le changement dans Label est immédiat.

VIII.Une calculatrice simple

Créez une nouvelle application et nommez par exemple votre projet Calculette.



Placez sur la fiche un Label qui simulera l'affichage à cristaux liquides des calculettes. Placez 10 boutons correspondant aux dix chiffres, ajoutez les boutons pour les quatre opérations, un ou deux boutons pour les corrections, un pour le changement de signe, un pour la virgule et évidemment une touche égal.

Il est possible de créer un bouton de la bonne taille et de le copier. Il reste alors à choisir pour chaque bouton, un nom (name) et son libellé (caption).

Modifiez certaines propriétés du label :
    Autosize à false
    Alignment à taRightJustify

Dans la suite les composants placés seront appelés :
    LabelAffichage
    BoutonZero, BoutonUn, BoutonDeux, BoutonTrois, BoutonQuatre, BoutonCinq, BoutonSix, BoutonSept, BoutonHuit, BoutonNeuf, BoutonPlus, BoutonMoins, BoutonMultiplier, BoutonDiviser, BoutonPlusMoins, BoutonVirgule, BoutonCorrection, BoutonEgal.

Les dix chiffres

A chaque fois que l'utilisateur choisira un chiffre, nous devons modifier la zone d'affichage en conséquence.

Une solution simple consiste à créer dix méthodes (procédures) pour répondre à l'événement "OnClick" de chaque bouton.

```
procedure TForm1.BoutonSeptClick(Sender: TObject);
begin
  LabelAffichage.Caption := LabelAffichage.Caption + '7';
end;
```

Il est cependant possible d'utiliser la même procédure pour répondre à l'événement "OnClick" de chaque bouton. Mais comment savoir alors s'il faut ajouter 1 ou 2 ou 3… ? C'est en utilisant la variable Sender. En effet, lorsqu'un événement est généré par un objet, la méthode associée reçoit en paramètre la variable Sender qui est un pointeur sur l'objet.

Le pointeur permet d'accéder aux propriétés de l'objet sur lequel il pointe.

Afin de préciser que l'objet est un bouton, nous devons indiquer au compilateur à l'aide du mot **as** que Sender est un pointeur sur un bouton (en non sur un objet quelconque).

Un objet est de type TObject alors qu'un bouton est de type TButton.

```
procedure TForm1.BoutonChiffre(Sender: TObject);
begin
  LabelAffichage.Caption := (Sender as TButton).Caption;
end
```

En fait, nous devons disposer d'une variable qui nous permettra de savoir si l'affichage doit être nettoyé ou s'il faut ajouter le chiffre à la fin. Enfin nous devons éviter les zéros au début du nombre.

Placez une variable de type booléen (boolean) dans la partie private de la classe, on pourra par exemple la nommer NouveauNombre.

```
procedure TForm1.BoutonChiffre(Sender: TObject);
begin
  if NouveauNombre then
  begin
    LabelAffichage.Caption := '';
    NouveauNombre := false;
  end;
  if LabelAffichage.Caption = '0' then
  begin
    LabelAffichage.Caption := (Sender as TButton).Caption;
  end
  else
  begin
    LabelAffichage.Caption := LabelAffichage.Caption + (Sender as TButton).Caption;
  end;
end;
```

Afin de pouvoir être connectée à un événement "OnClick", cette procédure devra être déclarée dans la classe, avant le mot **private**.

Pour connecter cette procédure à l'événement "OnClick" de chacun des 10 boutons chiffres, vous devez, dans le volet "Evénements" de l'inspecteur d'objet, choisir "BoutonChiffre" dans la liste des procédures compatibles avec cet événement. Vous ne devez donc pas faire un double clic pour créer une nouvelle procédure, vous devez au contraire choisir parmi celles qui existent.

Les touches opérateurs

Lorsque l'utilisateur choisit une des quatre touches correspondant aux quatre opérateurs, nous devons :

1. Effectuer l'opération en cours s'il y en a une.
2. Retenir le nombre actuellement affiché.
3. Retenir qu'il faudra commencer un nouveau nombre.
4. Retenir l'opération demandée.

Pour l'instant contentons-nous de créer une procédure vide "Calcule" qui se chargera plus tard d'effectuer l'opération en cours.

Nommons par exemple Pile une variable de type Extended (semblable au type real mais avec une meilleure précision) qui permettra de retenir le nombre affiché.

Voir l'aide en tapant sur F1 alors que le curseur est sur le mot Extended.

Codons par exemple avec le nombre 1 l'opérateur +, 2 l'opérateur -, 3 l'opérateur multiplier et 4 l'opérateur diviser.
Nommons par exemple Operateur (sans accent) une variable entière retenant ce code.

Voici la méthode pour le bouton plus. Faites de même pour les trois autres opérateurs.

```
procedure TForm1.Calcule;
begin
  // cette procédure n'est pas encore réalisée
end;

procedure TForm1.BoutonPlusClick(Sender: TObject);
begin
  Calcule;
  Pile := StrToFloat(LabelAffichage.Caption);
  NouveauNombre := true;
  Operateur := 1;
end;
```

StrToFloat permet de convertir une chaîne en un nombre décimal avec une précision importante (voir l'aide en tapant sur F1 alors que le curseur est sur le mot StrToFloat).

La touche "égal"

Il s'agit d'effectuer le calcul et d'initialiser la variable NouveauNombre.
Le calcul consiste à effectuer l'opération entre le nombre retenu dans la variable Pile et le nombre qui est à l'affichage en tenant compte de l'opération demandée. C'est le rôle de la procédure Calcule que nous allons terminer maintenant :

```
procedure TForm1.Calcule;
begin
  if Operateur = 1 then
  begin
    LabelAffichage.Caption :=
        FloatToStr(Pile + StrToFloat(LabelAffichage.Caption));
  end;
  if Operateur = 2 then
  begin
    LabelAffichage.Caption :=
        FloatToStr(Pile - StrToFloat(LabelAffichage.Caption));
  end;
  if Operateur = 3 then
  begin
    LabelAffichage.Caption :=
        FloatToStr(Pile * StrToFloat(LabelAffichage.Caption));
  end;
  if Operateur = 4 then
  begin
    LabelAffichage.Caption :=
        FloatToStr(Pile / StrToFloat(LabelAffichage.Caption));
  end;
  Operateur := 0;
end;
```

```
procedure TForm1.BoutonEgalClick(Sender: TObject);
begin
  Calcule;
  NouveauNombre := true;
end;
```

La touche Correction

Si l'utilisateur clique sur la touche de correction, nous devons oublier le nombre dans la variable Pile, oublier l'opérateur et mettre 0 à l'affichage:

```
procedure TForm1.BoutonCorrectionClick(Sender: TObject);
begin
  Pile := 0;
  Operateur := 0;
  LabelAffichage.Caption := '0';
end;
```

La touche virgule

Une virgule ou un point ? Cela dépend de la configuration de Windows. Pour que notre programme fonctionne dans tous les cas, nous pouvons utiliser la variable globale DecimalSeparator qui contient le caractère à utiliser.

```
procedure TForm1.BoutonVirguleClick(Sender: TObject);
begin
  if NouveauNombre then
  begin
    LabelAffichage.Caption := '0';
    NouveauNombre := false;
  end;
  LabelAffichage.Caption := LabelAffichage.Caption + DecimalSeparator;
end;
```

La touche +/-

Si l'affichage commence par le signe moins, nous devons l'enlever, sinon nous devons le mettre.
Il nous faut le moyen de comparer le premier caractère avec le symbole -. Nous pouvons utiliser la fonction Copy. Dans l'aide vous trouverez :

```
function Copy(S: string; Index, Count: Integer): string;
```

Dans notre cas Copy(LabelAffichage.Caption, 1, 1) correspond au premier caractère du Caption de LabelAffichage ou à la chaîne vide si le label est vide.

Pour supprimer le premier caractère nous pouvons également utiliser Copy avec 2 pour index et un grand nombre pour count afin de prendre tous les caractères restants.

```
procedure TForm1.BoutonPlusMoinsClick(Sender: TObject);
begin
  if LabelAffichage.Caption <> '0' then
  begin
    if Copy(LabelAffichage.Caption, 1, 1)='-' then
    begin
        LabelAffichage.Caption := Copy(LabelAffichage.Caption, 2, 255);
    end
    else
    begin
        LabelAffichage.Caption := '-' + LabelAffichage.Caption;
    end;
  end;
end;
```

Le programme n'est pas parfait

En effet :

- Nous ne traitons pas le cas de la division par zéro.
- L'utilisateur peut taper un nombre trop grand.
- Il peut également taper plusieurs virgules dans un nombre.
- L'utilisateur aimerait certainement pouvoir taper les chiffres au clavier.

La division par zéro peut être traitée avec **try** et **except**.

Nous pouvons limiter le nombre de caractères à l'aide de la fonction **length** qui permet de connaître le nombre de caractères d'une chaîne. Si ce nombre est trop grand, on n'effectue pas l'ajout du chiffre. L'utilisateur pourra toutefois multiplier plusieurs fois un très grand nombre par lui-même et finir par obtenir un message d'erreur.

Pour tester si une virgule existe déjà, on peut utiliser la fonction pos qui retourne la position d'une sous-chaîne dans une chaîne (et si la sous-chaîne n'existe pas, la fonction retourne 0).

Pour accepter la frappe des chiffres au clavier, il faut ajouter des méthodes pour répondre aux événements OnKeyPress, OnKeyDown de la fiche. On pourra aussi mettre à false la propriété TabStop de chaque bouton et mettre la propriété KeyPreview de la fiche à true.

IX. Un mini traceur de fonctions

Il s'agit de faire un programme capable de dessiner une représentation graphique à partir d'une formule de la forme y=f(x).

Propriété Canvas

Jusqu'à présent, nous n'avons pu écrire que dans un label ou un bouton.

La propriété Canvas (Canevas en français) offre une zone graphique sur laquelle vous pouvez écrire ou dessiner lors de l'exécution. La propriété Canvas s'applique en particulier aux PaintBox (boîtes à peindre).

Créez une nouvelle application et placez dans la fiche un objet PaintBox (vous le trouverez dans la palette d'outils "Système"). Dans la suite, je vais supposer que vous avez nommé cet objet "BoiteDessin" (sans accent).

Il va être possible d'écrire et de dessiner n'importe où dans BoiteDessin :

Pour écrire par exemple "Bonjour" en haut à gauche :

```
BoiteDessin.Canvas.TextOut(0,0,'Bonjour');
```

L'origine se trouve en haut à gauche de la boîte à peindre.

Pour placer un point rouge au milieu :

```
BoiteDessin.Canvas.Pixels[BoiteDessin.Width div 2,BoiteDessin.Height div 2]:=clRed;
```

Pixels est un tableau de points.
Div effectue une division entière alors que / (la barre oblique) retourne un nombre décimal qui ne conviendrait pas ici.

Pour tracer un segment allant du point placé à gauche et au milieu de la hauteur jusqu'au point placé à droite en haut :

```
BoiteDessin.Canvas.MoveTo(0,BoiteDessin.Height div 2);
BoiteDessin.Canvas.LineTo(BoiteDessin.Width,0);
```

MoveTo sert à se placer à l'endroit voulu mais aucun point n'est marqué, c'est un déplacement crayon levé.
LineTo sert à tracer un segment jusqu'au point précisé en paramètre.

Où placer ces lignes de programme ?

Si nous plaçons ces lignes de programme dans une procédure "OnClick" d'un bouton, le dessin sera effectué à chaque fois que nous ferons un clic sur le bouton.

Cette solution présente un défaut important : Si nous plaçons une fenêtre sur notre dessin puis l'enlevons, notre dessin sera perdu. Cette solution n'est pas acceptable dans un environnement de fenêtres comme Windows où l'utilisateur peut à tout moment exécuter plusieurs programmes en même temps.

A chaque fois qu'une "PaintBox" est créée ou recouverte puis découverte par un autre composant, Windows déclenche un événement "OnPaint".

En plaçant les lignes de programme qui permettent de réaliser notre dessin dans la procédure appelée lors de l'événement "OnPaint", notre dessin ne risque plus d'être effacé.

```
procedure TForm1.BoiteDessinPaint(Sender: TObject);
begin
  BoiteDessin.Canvas.TextOut(0,0,'Bonjour');
  BoiteDessin.Canvas.Pixels[BoiteDessin.Width div 2,BoiteDessin.Height
     div 2]:=clRed;
  BoiteDessin.Canvas.MoveTo(0,BoiteDessin.Height div 2);
  BoiteDessin.Canvas.LineTo(BoiteDessin.ClientWidth,0);
end;
```

Le point rouge au centre est petit et donc peu visible.

Tracé de f(x)=sin(x)

Pour chaque valeur de *x* dans l'intervalle correspondant à la boîte à peindre, nous allons :

- Calculer *y*.
- Faire un changement de variables afin de placer l'origine du repère au centre de notre "PaintBox" et obtenir ainsi un grossissement raisonnable.
- Placer les points. Cependant, afin de montrer qu'il s'agit d'une fonction continue, on pourrait, après avoir placé le premier point, tracer des segments.

Coordonnées mathématiques et informatiques

Appelons *x* et *y* les coordonnées mathématiques. Faisons varier *x* de -*?* à *?*.
Comme Delphi ne fait pas la distinction entre majuscules et minuscules, appelons *GX* et *GY* les coordonnées informatiques. *GX* va varier de 0 à BoiteDessin.Width.
Pour simplifier la formule appelons *W* le nombre BoiteDessin.Width et H le nombre BoiteDessin.Height..

L'abscisse mathématique varie de *2?* alors que l'abscisse informatique varie de *W*. Nous devons donc appliquer un coefficient de $\frac{2?}{W}$ pour passer des coordonnées informatiques aux coordonnées mathématiques et donc de $\frac{W}{2?}$ pour passer des coordonnées mathématiques aux coordonnées informatiques.
L'origine informatique est située en haut à gauche alors que nous allons placer l'origine mathématique au centre. Nous devons donc faire un décalage de $?\frac{W}{2}$ pour les abscisses informatiques, ce qui en abscisses mathématiques se traduit par un décalage de -*?*.

La formule pour passer de *GX* à *x* est donc :

$$x ? \frac{2?}{W} ? GX ? ?$$

Et la formule pour passer de *x* à *GX* est donc :

$$GX ? \frac{W}{2?} x ? \frac{W}{2}$$

Pour que la courbe ne soit pas déformée (repère orthonormal), nous devons appliquer le même coefficient pour les ordonnées. Le décalage informatique sera $\frac{H}{2}$. Le sens n'étant pas le même, la formule pour passer des coordonnées mathématiques aux coordonnées informatiques sera : $GY\ ?\ ?\ \frac{W}{2?}\, y\ ?\ \frac{H}{2}$

Choix des types pour les variables.

Pixels, MoveTo et LineTo ont besoin de nombres entiers (integer) alors que la fonction sin utilise des réels étendus (Extended). Les variables *x* et *y* seront donc de type extended alors que les autres seront de type integer.
Il est possible d'affecter une valeur entière à une variable réelle alors qu'il faut prendre la partie entière d'une valeur réelle pour l'affecter à une variable entière. En d'autres termes si i est entier et x réel étendu :

x := i	est correct
i := x	n'est pas accepté par le compilateur
i := round(x)	est correct (voir l'aide pour la fonction round).

Le programme

Vous avez maintenant les connaissances suffisantes pour créer la procédure qui sera exécutée lors de l'événement "OnPaint". Essayez de la faire sans regarder la solution proposée et comparez ensuite.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    BoiteDessin: TPaintBox;
    procedure BoiteDessinPaint(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.BoiteDessinPaint(Sender: TObject);
var x,y : extended;
var GX,GY : integer;
var W,H : integer;
begin
  W := BoiteDessin.ClientWidth;
  H := BoiteDessin.ClientHeight;
  for GX := 0 to W do
```

```
    begin
      x := 2*PI/W*GX - PI;
      y := sin(x);
      GY := round(-W/2/PI*y + H/2);
      BoiteDessin.Canvas.Pixels[GX,GY] := clRed;
    end;
  end;

  end.
```

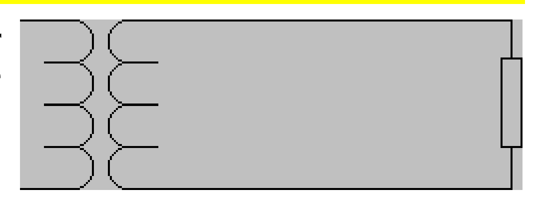
Idées d'exercices

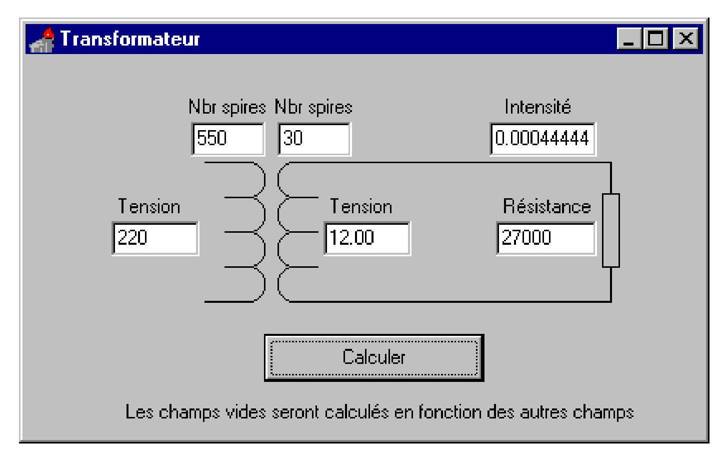
1. Améliorer le programme en ajoutant les axes et les graduations sur les axes.
2. Permettre à l'utilisateur de faire un "Zoom plus" et un "Zoom moins" sur la sinusoïde.
3. Permettre à l'utilisateur de faire un décalage vertical ou horizontal afin de montrer la courbe ailleurs qu'aux environs de (0,0).
4. Faire apparaître les coordonnées informatiques ou mieux mathématiques suivant la place du pointeur de souris.
5. Remplacer les points par des segments afin de montrer la continuité de la fonction.
6. Représenter plusieurs courbes avec des couleurs différentes pour mettre en évidence leurs points d'intersection.
7. Traiter le cas des fonctions qui ne sont pas toujours définies (en utilisant try et except).

X. Un transformateur et une résistance

Principe

A l'aide d'un logiciel de dessin, vous pouvez réaliser un schéma représentant un transformateur et une résistance.





Créez une nouvelle application et placez ce dessin sur la fiche.
Ajoutez des labels et des boîtes de saisie (composant TEdit).
Dans la suite les boîtes de saisie seront appelées :

EditTensionPrimaire
EditTensionSecondaire
EditNbrSpiresPrimaire
EditNbrSpiresSecondaire
EditResistance
EditIntensite

Le champ texte d'un composant TEdit est une chaîne de caractères, il sera nécessaire de transformer la chaîne en un nombre afin de pouvoir effectuer les calculs. Pour cela il sera pratique d'utiliser une variable numérique pour chaque composant TEdit.
Ces variables pourront être déclarées dans la partie privée de la classe :

```
TensionPrimaire : Extended;
TensionSecondaire : Extended;
NbrSpiresPrimaire : integer;
NbrSpiresSecondaire : integer;
Resistance : Extended;
Intensite : Extended;
```

La tension de l'enroulement primaire du transformateur, la tension de l'enroulement secondaire, le nombre de spires du primaire et le nombre de spires du secondaire sont des valeurs qui dépendent les unes des autres. Si trois valeurs sont connues, la quatrième peut être calculée.

Si la tension du secondaire est valide, l'intensité et la résistance sont des valeurs qui dépendent l'une de l'autre. Si l'une est connue, la deuxième peut être calculée.

Actions réalisées par le programme lorsque l'utilisateur clique sur le bouton "Calculer" :

1. Si la tension d'entrée n'est pas valide, elle est calculée (si impossible, on affiche un '?').
2. Si le nombre de spires du primaire n'est pas valide, il est calculé (si impossible, on affiche un '?').
3. Si le nombre de spires du secondaire n'est pas valide, il est calculé (si impossible, on affiche un '?').

4. Dans tous les cas le calcul de la tension à la sortie du secondaire est effectué (si impossible, on affiche un '?').
5. Si la valeur de la résistance n'est pas valide, elle est calculée (si impossible, on affiche un '?').
6. Dans tous les cas l'intensité est calculée (si impossible, on affiche un '?').

Pour forcer le calcul d'une valeur, l'utilisateur devra effacer le contenu de la zone de texte correspondante (ou y mettre des caractères incorrects ou une valeur négative) puis cliquer sur "Calculer".

La procédure val.

L'aide indique :

procedure Val(S; var V; var Code: Integer);

La procédure Val permet de transformer une chaîne en un nombre. Cette conversion n'étant pas toujours possible, un code d'erreur permet de savoir si cette conversion s'est bien passée.
Au retour de cette procédure, Code vaut le numéro du caractère qui a causé l'erreur, ou vaut 0 s'il n'y a pas eu d'erreur.
Exemple :

```
procedure essai;
var X:Extended;
var Code:integer;
begin
  val('12.456', X, Code);                    { essayer avec d'autres chaînes }
  if Code = 0 then
    ShowMessage('Conversion réussie')
  else
    ShowMessage('Le caractère au rang ' + IntToStr(Code) + ' n''est pas valide';
end;
```

Remarquez qu'entre then et else, il n'a pas été mis begin et end;
Ceci est possible lorsqu'une seule instruction doit être exécutée. Dans ce cas ne pas mettre de point-virgule avant le else puisque l'instruction de test n'est pas terminée.
Même remarque après le else.

La procédure str

L'aide indique :

procedure Str(X [: Width [: Decimals]]; var S);

La procédure Str permet de transformer un nombre réel en une chaîne.
Exemples d'utilisation de cette procédure :

Str(12.654, S);	Mettra ' 1.26540000000000E+0001' dans S	Affichage peu esthétique
Str(12.654:0, S);	Mettra ' 1.2E+0001' dans S	Le moins de caractères possible
Str(12.654:15, S);	Mettra ' 1.265400E+0001' dans S	chaîne de 15 caractères
Str(12.654:0:0, S);	Mettra '13' dans S	Aucun chiffre après la virgule
Str(12.654:0:3, S);	Mettra '12.654' dans S	Trois chiffres après la virgule.
Str(12.654:0:6, S);	Mettra '12.654000' dans S	Six chiffres après la virgule.

Dans la plupart des cas, on préférera utiliser la forme proposée dans les deux derniers exemples.

Solution

Ne regardez la solution proposée qu'après avoir cherché…

Certaines procédures se ressemblent, utilisez "Copie" et "Coller" afin de gagner du temps.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Image1: TImage;
    EditTensionPrimaire: TEdit;
    EditTensionSecondaire: TEdit;
    EditNbrSpiresPrimaire: TEdit;
    EditNbrSpiresSecondaire: TEdit;
    EditResistance: TEdit;
    EditIntensite: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    ButtonCalculer: TButton;
    procedure ButtonCalculerClick(Sender: TObject);
  private
    { Déclarations privées }
    TensionPrimaire:Extended;
    TensionSecondaire:Extended;
    NbrSpiresPrimaire:integer;
    NbrSpiresSecondaire:integer;
    Resistance:Extended;
    Intensite:Extended;
    procedure CalculeTensionPrimaire;
    procedure CalculeTensionSecondaire;
    procedure CalculeNbrSpiresPrimaire;
    procedure CalculeNbrSpiresSecondaire;
    procedure CalculeResistance;
    procedure CalculeIntensite;
  public
    { Déclarations publiques }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.CalculeTensionPrimaire;
var S:string;
begin
  if (TensionSecondaire<0) or (NbrSpiresPrimaire<0) or (NbrSpiresSecondaire<0) then
  begin
```

```
    EditTensionPrimaire.Text := '?';
    TensionPrimaire := -1;
  end
  else
  begin
    try
      TensionPrimaire := TensionSecondaire * NbrSpiresPrimaire
                                            / NbrSpiresSecondaire;
      if TensionPrimaire>=100 then
        Str(TensionPrimaire:0:0,S)
      else
        Str(TensionPrimaire:0:2,S);
      EditTensionPrimaire.Text := S;
    except
      EditTensionPrimaire.Text := '?';
      TensionPrimaire := -1;
    end;
  end;
end;

procedure TForm1.CalculeTensionSecondaire;
var S:string;
begin
  if (TensionPrimaire<0) or (NbrSpiresPrimaire<0) or (NbrSpiresSecondaire<0) then
  begin
    EditTensionSecondaire.Text := '?';
    TensionSecondaire := -1;
  end
  else
  begin
    try
      TensionSecondaire := TensionPrimaire * NbrSpiresSecondaire
                                            / NbrSpiresPrimaire;
      if TensionSecondaire>=100 then
        Str(TensionSecondaire:0:0,S)
      else
        Str(TensionSecondaire:0:2,S);
      EditTensionSecondaire.Text := S;
    except
      EditTensionSecondaire.Text := '';
      TensionSecondaire := -1;
    end;
  end;
end;


procedure TForm1.CalculeNbrSpiresPrimaire;
begin
  if (TensionPrimaire<0) or (TensionSecondaire<0) or (NbrSpiresSecondaire<0) then
  begin
    EditNbrSpiresPrimaire.Text := '?';
    NbrSpiresPrimaire := -1;
  end
  else
  begin
    try
      NbrSpiresPrimaire := round(TensionPrimaire * NbrSpiresSecondaire
                                                  / TensionSecondaire);
      EditNbrSpiresPrimaire.Text := IntToStr(NbrSpiresPrimaire);
    except
      EditNbrSpiresPrimaire.Text := '';
      NbrSpiresPrimaire := -1;
    end;
  end;
end;

procedure TForm1.CalculeNbrSpiresSecondaire;
begin
```

```
  if (TensionPrimaire<0) or (TensionSecondaire<0) or (NbrSpiresPrimaire<0) then
  begin
    EditNbrSpiresSecondaire.Text := '?';
    NbrSpiresSecondaire := -1;
  end
  else
  begin
    try
      NbrSpiresSecondaire := round(TensionSecondaire * NbrSpiresPrimaire
                                                      / TensionPrimaire);
      EditNbrSpiresSecondaire.Text := IntToStr(NbrSpiresSecondaire);
    except
      EditNbrSpiresSecondaire.Text := '';
      NbrSpiresSecondaire := -1;
    end;
  end;
end;

procedure TForm1.CalculeResistance;
var S:string;
begin
  if (TensionSecondaire<0) or (Intensite<0) then
  begin
    EditResistance.Text := '?';
    Resistance := -1;
  end
  else
  begin
    try
      Resistance := TensionSecondaire / Intensite;
      if Resistance>=100 then
        Str(Resistance:0:0,S)
      else
        Str(Resistance:0:2,S);
      EditResistance.Text := S;
    except
      EditResistance.Text := '?';
      Resistance := -1;
    end;
  end;
end;


procedure TForm1.CalculeIntensite;
var S:string;
begin
  if (TensionSecondaire<0) or (Resistance<0) then
  begin
    EditIntensite.Text := '?';
    Intensite := -1;
  end
  else
  begin
    try
      Intensite := TensionSecondaire / Resistance;
      if Intensite>=1 then
        Str(Intensite:0:3,S)
      else
        if Intensite>=0.001 then
          Str(Intensite:0:6,S)
        else
          Str(Intensite:0:9,S);
      EditIntensite.Text := S;
    except
      EditIntensite.Text := '?';
      Intensite := -1;
    end;
  end;
```

```
end;


procedure TForm1.ButtonCalculerClick(Sender: TObject);
var Code:integer;
begin
  val(EditTensionPrimaire.Text,TensionPrimaire,Code);
  if Code<>0 then TensionPrimaire:=-1;
  val(EditTensionSecondaire.Text,TensionSecondaire,Code);
  if Code<>0 then TensionSecondaire:=-1;
  val(EditNbrSpiresPrimaire.Text,NbrSpiresPrimaire,Code);
  if Code<>0 then NbrSpiresPrimaire:=-1;
  val(EditNbrSpiresSecondaire.Text,NbrSpiresSecondaire,Code);
  if Code<>0 then NbrSpiresSecondaire:=-1;
  val(EditResistance.Text,Resistance,Code);
  if Code<>0 then Resistance:=-1;
  val(EditIntensite.Text,Intensite,Code);
  if Code<>0 then Intensite:=-1;

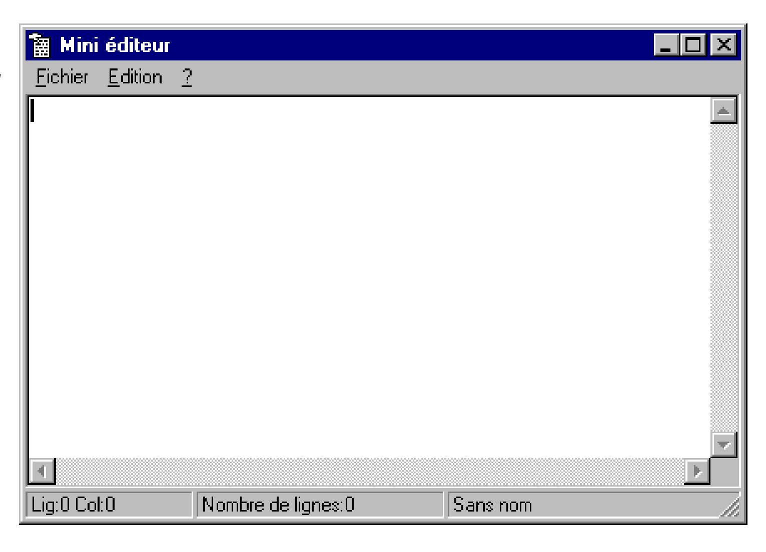
  if TensionPrimaire<0 then CalculeTensionPrimaire;
  if NbrSpiresPrimaire<0 then CalculeNbrSpiresPrimaire;
  if NbrSpiresSecondaire<0 then CalculeNbrSpiresSecondaire;
  CalculeTensionSecondaire;      { calculée dans tous les cas }

  if Resistance<0 then CalculeResistance;
  CalculeIntensite;              { calculée dans tous les cas }
end;

end.
```

XI. Un petit éditeur de textes

Ce chapitre permet de montrer l'utilisation :
- *du composant Mémo*
- *des menus*
- *des messages*
- *des boîtes de dialogue*
- *de plusieurs fiches*



Composant Memo

Créez un nouveau projet, placez un composant "Memo" (il s'appelle par défaut Memo1 et est de type TMemo), enregistrez votre unité et votre projet puis exécutez votre programme.
Vous constatez que le composant Memo permet de taper du texte, de sélectionner, de copier (avec Ctrl C) de coller avec (Ctrl V) de couper (avec Ctrl X) de défaire (Crtl Z).

Si vous modifiez la propriété "Align" du composant Memo1 en choisissant "alClient" toute la fenêtre est occupée par ce composant. A l'exécution, si l'utilisateur change la dimension de la fenêtre, la dimension du composant change automatiquement en même temps.

Un fichier texte est en général montré avec une police non proportionnelle, on choisira "Courier New" pour le composant Memo1.

Ajoutons un menu

Placez un composant "MainMenu" n'importe où sur la forme, puis faites un double clic dessus.
Ajoutez les menus et sous menus habituels :

```
&Fichier
    &Nouveau
    &Ouvrir
    &Enregistrer
    Enregistrer &sous
    -
    &Quitter
&Edition
    &Défaire
    &Couper
    Co&pier
    C&oller
&?
    &A propos
```

Vous remarquez que le symbole & a été utilisé, il permet de définir la lettre qui sera soulignée et qui permettra donc à l'utilisateur de choisir rapidement dans le menu à l'aide du clavier.
Le signe moins permet de créer une ligne séparatrice.
Vous pouvez fermer la fenêtre d'édition du menu.
Pour corriger ce menu par la suite, faites un double clic sur le composant placé sur votre forme et utilisez éventuellement les touches (entrée) (insère) (supprime).

Si vous exécutez ce programme maintenant, vous verrez le menu mais il est encore inefficace.

Menu Enregistrer simplifié

Pour simplifier nous allons supposer que le fichier texte s'appelle "c:\essai.txt" et que son nom se trouve dans la variable NomFich. Placez cette variable dans la partie private de TForm1 :

```
NomFich:string;
```

Pour enregistrer toutes les lignes de Meno1, on pourra utiliser la méthode SaveToFile définie par Delphi. L'aide donne :

```
procedure SaveToFile(const FileName: string);
```

En mode conception, cliquez sur le menu "Fichier" puis sur le sous menu "Enregistrer" pour créer la procédure attachée à l'événement clic de ce sous menu. Complétez cette procédure comme suit :

```
procedure TForm1.Enregistrer1Click(Sender: TObject);
begin
  NomFich:='c:\essai.txt';
  Memo1.Lines.SaveToFile(NomFich);
end;
```

Menu Ouvrir simplifié

Voir dans l'aide la procédure LoadFromFile.

```
procedure LoadFromFile(const FileName: string);
```

Voici une version simple de la procédure qui sera exécutée pour ouvrir un fichier texte:

```
procedure TForm1.Ouvrir1Click(Sender: TObject);
begin
  NomFich:='c:\essai.txt';
  Memo1.Lines.LoadFromFile(NomFich);
end;
```

Une variable pour se souvenir si le texte est modifié

Lorsque l'utilisateur est sur le point de perdre son document, il est habituel de lui demander s'il souhaite l'enregistrer avant. Il ne faut pas demander systématiquement mais seulement si le document a été modifié.
Il nous faut donc une variable booléenne pour retenir si le document a été modifié.

Créez par exemple la variable

```
Modif:boolean;
```

dans la partie private de TForm1
Cette variable sera automatiquement mise à false au démarrage du programme.
Cette variable devra être mise à true à chaque fois que l'utilisateur change le contenu de Meno1 (utilisez l'événement OnChange de Memo1).
Elle devra être mise à false :

- après un enregistrement réussi,
- lorsque l'utilisateur vient d'ouvrir un texte
- et lorsqu'il choisit de commencer un nouveau texte.

Permettre le choix du fichier à ouvrir

Pour l'instant le menu ouvrir ouvre toujours le fichier c:\essai.txt (une erreur se produit s'il n'existe pas). Nous allons placer n'importe où sur la forme un composant dialogue "OpenDialog".

Pour appeler la fenêtre habituelle permettant le choix du fichier à ouvrir, il suffira de faire :

```
OpenDialog1.execute;
```

ou mieux

```
if OpenDialog1.execute then
begin
  { ce qui doit être fait si l'utilisateur a validé son choix }
end;
```

Au retour, si l'utilisateur a validé son choix, le nom du fichier est placé dans

```
OpenDialog.FileName
```

```
procedure TForm1.Ouvrir1Click(Sender: TObject);
begin
  if OpenDialog1.execute then
  begin
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
    NomFich:=OpenDialog1.FileName;
    Modif:=false;
  end;
end;
```

Pour aider l'utilisateur à trouver les fichiers textes, modifiez la propriété Filter de OpenDialog1 :

Fichiers textes (*.txt)	*.txt
Tous (*.*)	*.*

Menu Enregistrer sous

Placez n'importe où sur la forme un composant "SaveDialog"

```
procedure TForm1.Enregistrersous1Click(Sender: TObject);
begin
  if SaveDialog1.execute then
  begin
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
    NomFich:=SaveDialog1.FileName;
    Modif:=false;
  end;
end;
```

Modifiez la propriété Filter de SaveDialog1 comme précédemment.

Améliorons la procédure du menu Enregistrer

Si NomFich est vide, c'est que l'utilisateur n'a pas encore choisi de nom pour son fichier. Nous devons en fait, faire comme s'il avait choisi "Enregistrer sous".

```
procedure TForm1.Enregistrer1Click(Sender: TObject);
begin
  if NomFich='' then
  begin
```

```
    Enregistrersous1Click(nil);
  end
  else
  begin
    Memo1.Lines.SaveToFile(NomFich);
    Modif:=false;
  end;
end;
```

La procédure Enregistrersous1Click nécessite un paramètre de type TObject, mais comme ce paramètre n'est pas utilisé dans la procédure, n'importe quel paramètre convient pour peu que son type soit accepté. Nil est un pointeur qui pointe sur rien.

Une fonction Fermer

Dans plusieurs cas, nous devons demander à l'utilisateur s'il accepte de perdre le document en cours. Il sera donc pratique de créer une fonction qui aura pour rôle de vider le composant Memo1 sauf si l'utilisateur décide d'abandonner la fermeture au dernier moment.

Principe

Tant que Memo1 n'est pas vide et que l'utilisateur ne désire pas abandonner nous devons, si le texte a été modifié, poser la question "Faut-il enregistrer le texte avant de le perdre ?".
Si l'utilisateur répond oui, nous appelons la procédure d'enregistrement.

> On sait que la procédure d'enregistrement a réussi en regardant la variable Modif. Si Modif est false, nous pouvons vider Memo1 et quitter la fonction Fermer avec succès.

Si l'utilisateur répond non, nous pouvons vider Memo1, mettre Modif à false et quitter la fonction Fermer avec succès.
Si l'utilisateur abandonne, nous ne vidons pas Memo1 mais nous quittons tout de même la fonction Fermer. Dans ce cas la fonction Fermer retourne false pour montrer que la fermeture a échoué.

Aide

- Le texte est vide si Memo1 n'a aucune ligne. Le nombre de lignes est donné par :
  `Memo1.Lines.Count`
  On peut aussi utiliser la propriété Text de Memo1. Text contient la totalité du texte contenu dans le composant Mémo, chaque chaîne autre que la dernière est terminée par les codes 13 et 10 correspondant au passage à la ligne (retour chariot).
- Pour indiquer la valeur de retour d'une fonction nous utiliserons result. Dans notre cas nous pouvons quitter cette fonction avec
  `result:=true`
  ou
  `result:=false`
- Utilisez MessageDlg pour poser la question.
- Utilisez l'instruction Case of pour envisager les trois cas.

Solution : Ne regardez le code qui suit qu'après avoir cherché !

```
function TForm1.Fermer:boolean;
var Abandon:boolean;
begin
  Abandon:=false;
  while (Memo1.Lines.Count<>0) and not Abandon do
  begin
    if Modif then
    begin
      case MessageDlg('Enregistrer ?',mtConfirmation,[mbYes,mbNo,mbCancel],0) of
        mrYes:
        begin
```

```
          Enregistrer1Click(nil);
          if not Modif then Memo1.Lines.Clear;
        end;
        mrNo:
        begin
          Memo1.Lines.Clear;
          Modif:=false;
        end;
        mrCancel:Abandon:=true;
      end;{case}
    end
    else
    begin
      Memo1.Lines.Clear;
    end;
  end;
  if Memo1.Lines.Count=0 then
  begin
    result:=true;
    NomFich:='';
  end
  else
  begin
    result:=false;
  end;
end;
```

Fichier Nouveau

Il suffit d'appeler la fonction Fermer sans même se préoccuper de sa valeur de retour.

Menu Quitter

Le mot close est suffisant pour quitter.

```
procedure TForm1.Quitter1Click(Sender: TObject);
begin
  Close;
end;
```

Lorsque la procédure close est appelée, elle génère un événement OnCloseQuery. Si nous attachons une procédure à cet événement, il nous sera encore possible de ne pas quitter.
Une idée serait de remplacer Close par :

```
      if Fermer then close;
```

Mais cette façon de faire ne convient pas. En effet, "Fichier" "Quitter" n'est pas la seule façon de quitter un programme Windows.

Evénement OnCloseQuery

Cet événement se produit lorsque l'utilisateur est sur le point de quitter. Pour quitter, l'utilisateur peut utiliser "Fichier" "Quitter", la procédure close est alors appelée, mais il peut aussi cliquer sur la case quitter (en haut à droite) ou utiliser le menu système (en haut à gauche) ou faire Alt F4…
Dans tous les cas l'événement OnCloseQuery se produit. Il est encore possible de ne pas quitter.
Voici la procédure attachée à cet événement :

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin

end;
```

En mettant dans cette procédure

```
CanClose:=false;
```

le programme ne s'arrête plus. Si vous voulez faire l'essai, il est prudent d'enregistrer votre projet avant. Vous pouvez toutefois arrêter votre programme en mettant un point d'arrêt sur la ligne située après la ligne contenant CanClose, faites "Fichier" et "Quitter". A l'aide de la fenêtre "Evaluation / modification" (Ctrl F7 lorsque le curseur est sur le mot CanClose) mettez à true la variable CanClose. Tapez F9 pour terminer correctement votre programme.

En fait, CanClose ne devra être mise à false que si la fonction Fermer échoue.

```
procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if not Fermer then CanClose := false;
end;
```

Edition Défaire

Comme le composant Memo1 sait déjà "Défaire", "Copier", "Couper" et "Coller", il suffit de lui envoyer un message pour demander ce travail. En regardant dans l'aide on trouve :

```
LRESULT SendMessage(
    HWND hWnd,       // handle of destination window
    UINT Msg,        // message to send
    WPARAM wParam,   // first message parameter
    LPARAM lParam    // second message parameter
   );
```

Il s'agit de la déclaration en langage C de la fonction SendMessage. Parmi les quatre paramètres seuls les deux premiers nous intéressent ici.
hWnd est le "handle" de l'objet à qui nous allons envoyer le message. Le handle est un numéro (un integer) qui permet d'accéder à un objet.
Lorsqu'un objet est créé, il est placé en mémoire. Windows se réserve le droit à tout moment de déplacer les objets dans sa mémoire. Il n'est donc pas question, pour l'utilisateur d'accéder directement à un objet comme on pouvait le faire avec les "vieux" programmes dos. Le handle est attribué par Windows lors de la création de l'objet et ne changera pas pendant toute la durée de vie cet objet.
Le handle de Memo1 est obtenu par :

```
Memo1.Handle
```

Msg est un numéro (un integer) de message. Pour avoir une liste des messages utilisés par Windows vous pouvez regarder le fichier Message.pas par exemple avec le bloc-notes. Si Delphi 3 a été installé dans le répertoire habituel sur le disque C, voici le chemin à utiliser pour ouvrir ce fichier :

```
c:\Program Files\Borland\Delphi 3\Source\Rtl\Win\Messages.pas
```

Vous trouverez par exemple que WM_UNDO correspond à $304.

Pour défaire la dernière action faite par l'utilisateur dans Memo1, nous pouvons utiliser :

```
SendMessage(Memo1.Handle,WM_UNDO,0,0);
```

Memo1 recevra le message WM_UNDO et exécutera la procédure correspondante.

Edition Copier, Couper, Coller

Il est possible d'utiliser également SendMessage avec :

```
WM_COPY, WM_CUT et WM_PASTE.
```

Il est cependant plus facile d'utiliser les méthodes

```
CopyToClipboard;
CutToClipboard;
PasteFromClipboard;
```

Vous trouverez ces méthodes dans l'aide de TMemo.

A propos

Beaucoup d'applications Windows possèdent une boîte de dialogue souvent nommée "A propos", qui donne le nom du programme, sa version, le nom de l'auteur…
Faites "Fichier" et "Nouvelle Fiche" (avec Delphi 3 vous pouvez aussi faire "Fichier" "Nouveau" "Fiches" et "Boîte à propos").
Nommez cette forme par exemple Bap et enregistrez l'unité en l'appelant par exemple UBap
Améliorez cette fiche…
Il faut au moins un bouton "Ok" ou "Fermer" qui aura pour rôle de fermer cette fenêtre facilement.
Pour l'appeler à partir du menu "A propos" de la forme principale, vous pouvez attacher la procédure suivante au sous menu "A propos".

```
Bap.Showmodal;
```

L'utilisateur devra quitter la boîte "A propos" pour pouvoir continuer à utiliser le mini éditeur.

Idées d'améliorations

- Si l'utilisateur ne met pas d'extension, ajoutez l'extension txt
  La fonction suivante ajoute .txt si nécessaire :

```
function TForm1.AjouteEventuellementExtension(NF:string;Ext:string):string;
begin
  if UpperCase(ExtractFileExt(NF))='' then
    result:=NF + Ext
  else
    result:=NF;
end;
```

  Par exemples
  `AjouteEventuellementExtension('Bonjour','.txt')` retournera `'Bonjour.txt'`
  `AjouteEventuellementExtension('Bonjour.abc','.txt')` retournera `'Bonjour.abc'`
  `AjouteEventuellementExtension('Bonjour.','.txt')` retournera `'Bonjour.'`
  On utilisera cette fonction avant de faire appel à SaveToFile

- La variable Modif n'est pas nécessaire pour un composant Mémo car ce composant possède déjà la variable Modified.

- Dans le menu, utilisez la propriété "Short Cut" pour Enregistrer, Défaire, Copier, Couper et Coller en choisissant respectivement Ctrl S, Ctrl Z, Ctrl C, Ctrl X, Ctrl V.

- Ajoutez un sous menu "Rechercher" dans le menu "Edition". Vous pouvez profiter du composant FindDialog.
  Pos donne la position d'une sous chaîne dans une chaîne.

- Remplacez Memo1 par un composant RichEdit et il vous sera facile d'imprimer.

- Ajoutez un composant StatusBar pour y mettre le numéro de ligne et de colonne.

XII. Liaison série

Ce chapitre suppose que vous avez le composant CommPortDriver qui gère la série. Vous pouvez le télécharger à l'adresse suivante :

ftp://dialog.ac-reims.fr/pub/Delphi/Serie32/Serie32.ZIP

Si vous avez réalisé le petit éditeur du chapitre précédent, vous pouvez le compléter en ajoutant les menus suivants :

&Série
&Configurer
&Envoyer
&Recevoir
&Arrêter de recevoir

Sinon, vous pouvez placer un composant Mémo et quelques boutons pour remplacer le menu.

Ce composant permet d'envoyer et de recevoir en même temps. Nous allons nous contenter de faire l'un ou l'autre.

Le composant CommPortDriver

En mode conception :
- *Installer le composant s'il n'est pas encore installé*
- *Placer un composant sur votre forme*
- *Modifier éventuellement les propriétés*

En mode exécution :
- *Choisir le port et le configurer*
- *Utiliser la méthode Connect*
- *Envoyer et/ou recevoir*
- *Utiliser la méthode Disconnect*

Choisir le port et configurer la série

Ceci pourrait être fait en mode conception mais l'utilisateur ne pourrait pas changer le paramétrage. Nous allons préférer lui donner le choix.

Créez une nouvelle fiche (pour la suite je considère que vous l'avez nommée FormSerie).
Placez les composants comme indiqué ci-contre :
Eventuellement un composant GroupBox.
Un composant CommPortDriver.
Cinq composants Label.
Cinq composants ComboBox (listes déroulantes).
Trois boutons.

J'ai mis bsDialog dans la propriété BorderStyle de la forme.
J'ai mis biMinimize et biMaximize à false dans la propriété BorderIcons de la forme.



Nous allons remplir les listes déroulantes avec les valeurs autorisées.

Je suppose dans la suite que vous avez nommé la liste déroulante du choix du port "ComboBoxPort". J'explique ce qui concerne le choix du port, vous pourrez faire de même pour les autres paramètres.
Entrez les valeurs autorisées dans la propriété Items de ComboBoxPort soit COM1 sur la première ligne, COM2 sur la deuxième ligne… jusqu'à COM4 (Windows autorise jusqu'à 16 ports mais les PC n'en ont souvent que 2).
Dans la procédure attachée à l'événement OnCreate, nous allons activer le port choisi lors de la conception.
Voici la partie de la procédure FormCreate qui concerne ComboBoxPort :

```
  case CommPortDriver1.ComPort of
    pnCOM1:ComboBoxPorts.ItemIndex:=0;
    pnCOM2:ComboBoxPorts.ItemIndex:=1;
    pnCOM3:ComboBoxPorts.ItemIndex:=2;
    pnCOM4:ComboBoxPorts.ItemIndex:=3;
  else
    ComboBoxPorts.ItemIndex:=-1;
  end;{case}
```

Lorsque l'utilisateur choisit le bouton "Appliquer" ou le bouton "Ok", nous devons prendre en compte ses choix. Pour ComboBoxPort cela donne :

```
  case ComboBoxPorts.ItemIndex of
    0:CommPortDriver1.ComPort:=pnCOM1;
    1:CommPortDriver1.ComPort:=pnCOM2;
    2:CommPortDriver1.ComPort:=pnCOM3;
    3:CommPortDriver1.ComPort:=pnCOM4;
    else
      { prévenir qu'il y a une erreur }
  end;{case}
```

Envoi vers la série

Utilisez la méthode Connect de CommPortDriver puis au choix SendByte ou SendChar ou SendString. Pour terminer utilisez Disconnect.

```
procedure Envoyer(S:string);
begin
  if CommPortDriver1.Connect then
   begin
    CommPortDriver1.SendString(S);
    CommPortDriver1.Disconnect;
  end;
end;
```

Réception de la série

Lorsque des caractères arrivent sur la série, l'événement OnReceiveData se produit. Le nombre de caractères reçus est connu grâce à DataSize. Les caractères reçus sont placés dans une zone mémoire pointée par DataPtr. Pour obtenir le premier caractère nous pouvons utiliser PChar(DataPtr)[0], pour le deuxième PChar(DataPtr)[1]…
Nous pouvons par exemple ajouter ces caractères à une chaîne de caractères, ou l'afficher dans un composant quelconque. J'ai choisi de les envoyer dans le composant Memo1.

Dans la partie private de la forme, déclarez une variable nommée par exemple Hdle de type tHandle :

```
    Hdle:tHandle;
```

Lorsque l'utilisateur décide d'activer la réception, il faut prévoir où vont arriver les caractères puis "ouvrir la série" avec Connect.

Dans le menu Recevoir de la forme principale :

```
procedure TForm1.Recevoir1Click(Sender: TObject);
begin
  FormSerie.Recevoir(Memo1.Handle);
end;
```

Dans la forme FormSerie :

```
procedure TFormSerie.Recevoir(h:THandle);
begin
  Hdle:=h;
  CommPortDriver1.Connect;
end;
```

Lorsque l'événement OnReceiveData se produit :

```
procedure TFormSerie.CommPortDriver1ReceiveData(Sender: TObject;
  DataPtr: Pointer; DataSize: Integer);
var i:integer;
var p:pointer;
begin
  for i:=0 to DataSize-1 do
  begin
    SendMessage(Hdle,WM_Char,ord(PChar(DataPtr)[i]),0);
  end;
end;
```

Idées d'améliorations

- Créez une zone pour la réception et une autre pour l'envoi.
- Ne pas envoyer une chaîne, mais un caractère à la fois afin de permettre à l'utilisateur de faire quelque chose pendant l'envoi (voir l'aide sur la procédure ProcessMessages ou utilisez un Timer).